F3

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 403 124 A2**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: **90305974.9**

(51) Int. Cl.5: **G06F 12/08**

(22) Date of filing: **31.05.90**

(30) Priority: **16.06.89 US 367838**

(43) Date of publication of application:
19.12.90 Bulletin 90/51

(84) Designated Contracting States:
DE FR GB

(71) Applicant: **International Business Machines Corporation**
Old Orchard Road
Armonk, N.Y. 10504(US)

(72) Inventor: **Sherman, Arthur Michael**
15920 La Escuela
Morgan Hill, CA 95037(US)
Inventor: **Walling, Lonnie Scott**
9780 Higway 9
Ben Lomond, CA 95005(US)

(74) Representative: **Blakemore, Frederick Norman**
IBM United Kingdom Limited Intellectual Property Department Hursley Park
Winchester Hampshire SO21 2JN(GB)

(54) Overlay swapping.

(57) Overlay swapping is enabled by differentiating Overlays each into a code portion and into a data portion, storing the data portions within a CPU directly-accessible memory, storing the code portions within a memory not-directly accessible by the CPU, and swapping a required code portion into the directly-accessible memory for execution by the CPU such that the swapped-in code portion has access to a plurality of the data portions stored within the directly-accessible memory.

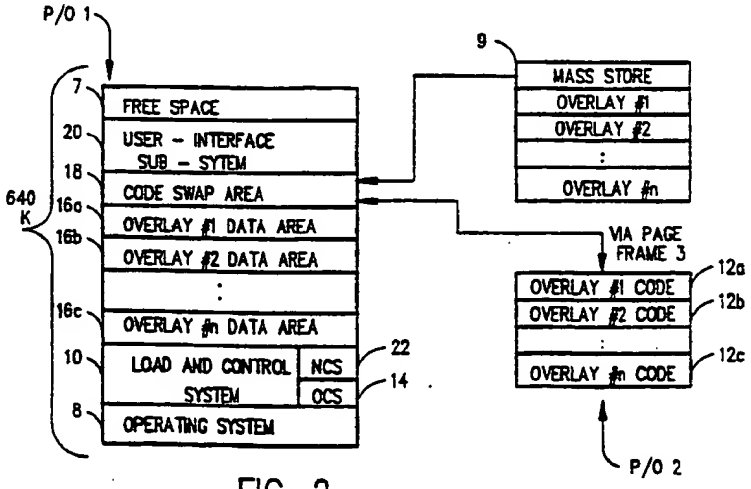

FIG. 2

EP 0 403 124 A2

This invention relates to overlay swapping in data processing apparatus.

One problem currently encountered in data processing systems having CPU-constrained main memories is that of calling one overlay from another when the data concerned is common to both overlays. This has been determined to arise from the inability of multiple code overlays to swap into and out of a single area within RAM while each overlay also maintains a non-swappable data area within the RAM. Were it possible to separate code and data, with the code being swappable and the data operated on by the code being non-swappable, it would allow a given procedure within a code swappable overlay to be invoked from anywhere within the system, including other code swappable overlays.

The ability for one swappable overlay to invoke a procedure from another swappable overlay should provide a number of benefits. Typically, because a calling overlay is completely swapped out by the called overlay, the called overlay has no access to any data within the caller, including the caller's stack. This situation severely limits the type of procedures that can be added to an overlay.

The present invention provides a data processing system including memory and I/O, supporting a spectrum of system addresses to memory and the paging of information with respect to memory, and provided with initialisation logic to reserve a paging window in the spectrum of system addresses for paging overlay code and residence segments of memory at other system addresses for overlay data.

From another aspect, the present invention provides a method of swapping overlays in a data processing system including a paging memory, supporting a spectrum of system addresses to memory and the paging of information with respect to memory, the method comprising reserving a paging window in the spectrum of system addresses for paging overlay code and maintaining overlay data resident in segments of memory at other system addresses.

The above enables the managing the swapping of code associated with overlays into and out of an area within a main memory while maintaining non-swappable data for each of the overlays within another area of the main memory.

This is regarded as being a practical way to provide, in a data processing system having a CPU-constrained main memory, a method whereby a swappable overlay is enabled to invoke a procedure from another swappable overlay.

Disclosed hereinafter is a main memory constrained data processing system a process in which multiple code overlays are swapped into and out of a single area within the main memory while each overlay also maintains a non-swappable data area within the main memory. The called overlay that is swapped into a main memory has access to data associated with a calling, swapped out overlay, including the calling overlay's stack.

This disclosed method of managing the storage of overlays before the execution thereof, is for use in a data processing system having a CPU coupled to a first, directly accessible main memory, the main memory storing code and data associated with the code. A particular group of code statements and an associated data portion comprise an overlay.

The disclosed method includes the steps of differentiating the overlays into a code portion and into a data portion, storing the data portions within the first directly accessible memory, storing the code portions within a second memory that is not-directly accessible by the CPU, and swapping a required code portion into the first memory for execution by the CPU such that the swapped-in code portion has access to a plurality of the data portions stored within the first memory.

In greater detail, the method includes the steps of

(a) allocating a code area within the main memory having a number of storage locations sufficient to accommodate a largest executable code portion associated with one or more required overlays,

(b) allocating one or more data areas within the main memory each of which has a number of storage locations sufficient to accommodate one data portion associated with one of the required overlays, and

(c) allocating one or more areas within an expanded memory, each of the one or more areas having a number of storage locations sufficient to accommodate a code portion of one of the required overlays.

The method further includes the steps of, for each of the required overlays,

(d) storing the data portion within the allocated data area,

(e) storing the code portion within the allocated code area,

(f) relocating within the stored code portion any references to data such that the references reflect the data storage location within the associated allocated data area, and

(g) copying the stored and relocated code portion to the allocated area within the expanded memory.

The present invention will be described further by way of example with reference to an embodiment thereof as illustrated in the accompanying drawings, in which:-

Fig. 1 is a diagram of the memory organisa-

tion of a CPU-constrained main memory having a expanded memory portion that is swappable into a region of the main memory; and

Fig. 2 shows in, greater detail, that organisation.

Fig. 1 shows an Expanded Memory System (EMS). The system includes a one megabyte (1024K) physical memory 1 address space and a typically larger, for example up to 32 megabytes, expanded memory 2 address space. The physical memory 1 is coupled to and is directly accessible by a CPU that reads and writes the memory 1. The expanded memory 2 provides additional memory beyond a maximum addressable memory limit of the CPU. For example, microprocessors such as the 8086, 8088 and the 80286, operated in the real mode, are capable of physically addressing up to 1024K bytes of memory. The expanded memory is therefore accessed through a page frame 3 "window", or predetermined range of addresses, located within the directly accessible 1024K physical address range. In the example shown the page frame 3 occupies the range of addresses between 768K and 960K but this range of addresses need only be 64K bytes. In other embodiments the page frame 3 can occupy other ranges of addresses. The expanded memory 2 is partitioned into segments referred to as logical pages 4 each of which is typically 16K bytes in length. The logical pages 4 are accessed through the physical block of memory associated with the page frame 3. The page frame 3 contains a plurality, typically four, directly accessible physical pages 5, each of the physical pages 5 also being typically 16k bytes in length. In the disclosed arrangement only one physical page 5 is used. It should be noted that the physical page(s) 5 can be mapped at address locations other than those illustrated, such as within the lower 640K of memory.

In operation the EMS maps a logical page 4 of expanded memory 2 into a particular one of the physical pages 5 of the page frame 3, thereby providing the CPU direct access to the logical page 4. The operation of one particular type of EMS that is particularly suitable for use with the disclosed arrangement is set forth in a publication "LotusTM/IntelTM/MicrosoftTM Expanded Memory Specification Version 4.0", dated October 25, 1987, and commonly referred to as LIM EMS. Lotus is a trademark of Lotus Development Corporation, Cambridge, MA, Intel is a trademark of Intel Corporation, Hillsboro, OR and Microsoft is a trademark of Microsoft Corporation, Redmond, WA.

The page frame 3 may be located above an address of 640K, that is, above a region 6 that is available for use by the operating system and application programs. The page frame 3 is typically located within a region having addresses predefined to contain video and other types of adapters. However, as was previously stated the page frame 3 may be located at other positions within the physical memory 1 address space.

It should be realised that EMS techniques other than LIM EMS are currently known and that the teaching of the disclosed arrangement is not to be construed to be limited to operation with only one particular type of EMS or with any one particular type of operating system. As will be made apparent the teaching of the disclosed arrangement is applicable for use in any system that employs paged access to memory not directly accessible by a CPU.

Figure 2 illustrates in greater detail the operation of the disclosed arrangement and in particular shows the regions 6 and 7 of the physical memory space 1. At locations above the Operating System (OS) 8 a Load and Control System 10 is resident and determines which program modules, or Overlays, typically stored on a mass store 9 such as a disk, are to be loaded in support of a particular application program being run. This information is obtained from a Load List associated with an application, the Load List including a list of required Overlays. Each Overlay typically includes an executable code portion and a data portion.

It should be noted that as used herein the term "Overlay" is to be interpreted both in a traditional sense to mean a code Overlay, such as a module having a .OVL extension, as well as in a broader sense to mean a complete and independent program. As such, the disclosed arrangement provides a traditional code Overlay with the ability to call and return from a second Overlay. Also, the disclosed arrangement provides a capability for an entire program to be used as a swappable Overlay by separately maintaining the data portion of the program. It should be noted that the Overlays are typically in a form associated with files known as .EXE files, a file type having a structure known in the art. The structure of .EXE files is set forth in a publication entitled "Disk Operating System Technical Reference 6138536", February 1985, published by the IBM Corporation, Chapter 10, "EXE File Structure and Loading", the disclosure of which is incorporated herein by reference. Overlays may also be associated with extended .EXE file structures of a type employed with OS/2.

Space for a data structure, specifically an Overlay Control Structure (OCS) 14, is allocated by the Load and Control System 10 to maintain a record of a current state of all of the Overlays available to the system. The Overlay Control Structure 14 includes the following fields of data.

Overlay Status.

These fields include information related to whether a particular overlay is swappable or resident and, for example, if the Overlay is swappable whether the Overlay is currently loaded within a Code Swap Area 18(to be described).

Overlay Name.

These fields contain the file names associated with the Overlays so that the Overlays can be located on the mass store 9.

Overlay Initialisation Entry Point.

These fields contain the initialisation entry point address associated with each Overlay. This entry point is employed when a swappable Overlay is first loaded and initialised, as will be described below.

Overlay Primary Entry Point.

These fields contain an address of the functional entry point for each Overlay.

Overlay Termination Entry Point.

These fields contain an address of a de-initialisation entry point for each Overlay.

Overlay LIM Handle.

These fields contain LIM handle information as specified by the LIM EMS.

Overlay LIM Page Count.

These fields contain the number of 16K byte pages associated with each of the Overlay code segments stored within the expanded memory 2.

Overlay Resident Data Segment Address.

These fields contain the address of the data segments associated with the data areas 16.

Overlay Code Size in Paragraphs.

These fields contain the required size of each of the code segments in "paragraphs", or 16 byte increments. The values contained within these fields are employed to size the Code Swap Area 18 such that a maximum of 15 bytes of unused storage is allocated for the largest code portion.

In the disclosed arrangement two passes are made over the required swappable Overlays stored on mass store 9. During a first pass the Load and Control System 10 determines a size of a data portion and an executable code portion for each of the specified Overlays. The code segment and data segment information is available for .EXE programs at predetermined positions within the file. An amount of storage within main memory 1 that is required to store the largest Overlay code portion is also determined while performing this first pass size determination and is retained. The size information is stored in the Overlay Control Structure 14 within the Overlay Code Size in Paragraphs field.

In the disclosed arrangement, separate areas (16a, 16b, 16c) are allocated in memory for the storage of the data associated with the Overlay code portions and a separate area is allocated for the code portion of a currently swapped-in Overlay. The region of memory allocated for storage of the Overlay code portion is the Code Swap Area 18, the Code Swap Area 18 having enough locations to store the largest of the Overlay code portions, in paragraphs, as previously determined. The segment addresses of the data areas 16 are stored in the Overlay Control Structure 14 within the Overlay Resident Data Segment Address fields, as previously described.

Separate areas 12a, 12b and 12c are allocated within the Expanded Memory 2 for each Overlay code portion. The size in pages and the LIM-related handle information for each of the code areas 12 is stored in the Overlay Control Structure 14 as previously described.

Next a second pass is run that loads by segments each Overlay identified as being required for the application. The code segments associated with the .EXE files on the mass store 9 are individually loaded into the code swap area 18. This loading is done independently of the Operating System 8 by the Load and Control System 10 according to the procedure set forth below.

From information contained within each required .EXE file the data segment(s) and code segments are identified, as is the information necessary to relocate the data segments. The data portion of a specified Overlay is copied into the pre-allocated area 16 within the memory 1. The code segments of the specified Overlay are individually loaded into the Code Swap Area 18. Any references to data within a particular code segment are re-located as a function of the address of the data segment of the associated data area 16.

The Overlay Initialisation Entry Point, Overlay Primary Entry Point, and an Overlay Termination Entry Point are retrieved from the Overlay and stored within the corresponding data fields within the Overlay Control Structure 14. In addition, the Overlay Initialisation Entry Point is called to perform any required Overlay initialisation and to store the address of the Load and Control System 10 within the Overlay.

The relocated and initialised Overlay code is written to the window 3 and then swapped into the Expanded memory 2 at the pre-allocated LIM area 12.

After so initialising all of the required Overlays control is subsequently passed to a User Interface Sub-System 20. This component invokes requested functions within the system by calling the Load and Control System 10 and providing the name of the function to be invoked and the parameters, if any, to be passed to the function.

Responsive to a request by the User Interface Sub-System 20 the Load and Control System 10 first checks the Overlay Control Structure 14 Overlay Status fields to determine if the Overlay containing the called function is a swappable or a resident Overlay and, if the Overlay is determined to be swappable, if the Overlay is currently loaded within the Code Swap Area 18. If the Overlay having the called procedure is not currently loaded the Load and Control System 10 swaps the Overlay code portion into the window 3 from the associated code area 12 and subsequently moves the code into the Code Swap Area 18 for execution. The Load and Control System 10 also updates the Overlay Control Structure 14 to indicate that the required Overlay now occupies the Code Swap Area 18. It should be noted that, in disclosed arrangement, the data portion of this particular Overlay is already resident within one of the data areas 12 and that the data portion of the Overlay swapped out of the Code Swap Area 18 is also still resident.

The Load and Control System 10 subsequently passes control to the Overlay Primary Entry Point. The Overlay thereafter calls the specified procedure within the Overlay in a conventional manner.

In the disclosed arrangement, a call can be made from within a first swappable Overlay to a function within a second swappable Overlay. The following method accomplishes this action.

The Load and Control System 10 is invoked by the first Overlay through the User Interface Sub-System 20. Through the Overlay Control Structure 14 the Load and Control System 10 determines that both the calling Overlay and the called Overlay are swappable Overlays. A return address on a stack of the caller, the stack being located within the associated data area 16, is modified by the Load and Control System 10 to indicate an address within the Load and Control System 10. This is done to ensure that when the called function terminates the Load and Control System 10 is enabled to intervene to reload the first Overlay. The Load and Control System 10 also maintains the first Overlay identification and the original return address in a push-down stack referred to as a Nesting Control Structure (NCS) 22. The size of the Nesting Control Structure 22 determines a maximum number of nesting levels, that is, how many swappable Overlays calling other swappable Overlays can occur before the Nesting Control Structure 22 overflows. The Nesting Control Structure 22 is employed by the Load and Control System 10 to re-load the original calling Overlay before returning control to it at the return address.

As an example, an application has a plurality of swappable Overlays associated therewith, including a first and a second Overlay. The second Overlay includes I/O functions, such as a function that receives input characters from a source. The first Overlay includes a character buffer within the associated data area 16. The first Overlay invokes the character input function of the second Overlay, passing as a stacked parameter the data segment address of the character buffer. The second Overlay is swapped in, as previously described, and the character input function is invoked and executed to place characters into the buffer of the first Overlay. After the second Overlay terminates execution the first Overlay is swapped back in, using information from the NCS 22, by the Load and Control System 10 and continues execution. As can be seen, the disclosed arrangement makes possible such an interaction by providing that the data portion of the first swappable Overlay remain resident within CPU-accessible memory while the first Overlay is swapped out to EMS and during the time that the second Overlay is swapped in and is executing.

The foregoing implies a data processing system including memory and I/O, supporting a spectrum of system addresses to memory and the paging of information with respect to memory, and provided with initialisation logic to reserve a paging window in the spectrum of system addresses for paging overlay code and residence segments of memory at other system addresses for overlay data making possible the swapping of overlays by reserving a paging window in the spectrum of system addresses for paging overlay code and maintaining overlay data resident in segments of memory at other system addresses.

Claims

1. A data processing system including memory

and I/O, supporting a spectrum of system addresses to memory and the paging of information with respect to memory, and provided with initialisation logic to reserve a paging window in the spectrum of system addresses for paging overlay code and residence segments of memory at other system addresses for overlay data.

2. A data processing system having a first memory directly addressable by a CPU supporting a spectrum of system addresses and a second memory not directly addressable by the CPU means but swappable with respect to the first memory, the system including one or more Overlays, each having a code portion and a data portion, and executing the Overlays by the steps of:
differentiating the Overlays into a code portion and into a data portion;
storing the data portions within the first memory means;
storing the code portions within the second memory means; and
swapping a required one of the code portions into the first memory means for execution by the CPU means.

3. A system as claimed in claim 2, in which initialisation includes allocating storage within the first memory means for a data structure having fields descriptive of the Overlays.

4. A system as claimed in claim 3, wherein information is stored within the data structure for each of the Overlays, such information including:
the status indicative of at least whether a code portion of a specific Overlay is currently swapped into the first memory means;
the name associated with the Overlay;
the Overlay initialisation entry point address;
the Overlay functional entry point address;
the Overlay termination entry point address;
information indicative of a number of storage locations within the second memory means required to store the code portion; and information indicative of where the data portion of the Overlay is stored within the first memory means.

5. In a data processing system having code statement execution means coupled to main memory means for storing code statements and data associated with the code statements, a particular group of code statements and an associated data portion comprising an Overlay, a method of managing the storage of Overlays prior to execution thereof comprising the steps of:
allocating a code area within the main memory means having a number of storage locations sufficient to accommodate a largest code portion associated with one or more required Overlays;
allocating one or more data areas within the main memory means each of which has a number of storage locations sufficient to accommodate the data portion associated with one of the required Overlays;
allocating one or more code areas within an expanded memory means, each of the one or more code areas having a number of storage locations sufficient to accommodate the code portion of one of the required Overlays; and
for each of the required Overlays,
storing the data portion within the allocated data area;
storing the code portion within the allocated code area;
relocating within the stored code portion any references to data such that the references reflect the data storage location within the associated allocated data area; and
storing the relocated code portion into the allocated code area within the expanded memory means.

6. A method as claimed in claim 5 and including an initial step of allocating storage within the main memory means for an Overlay Control Structure.

7. A method as claimed in claim 6 and including an additional step of retrieving specific data from the Overlay and storing the specific data in the Overlay Control Structure, the specific data including:
an Overlay Initialisation Entry Point;
an Overlay Primary Entry Point; and
an Overlay Termination Entry Point.

8. A method as claimed in claim 7 and including a step of, for each of the allocated data areas, storing an address of the allocated data area within the Overlay Control Structure.

9. A method as claimed in claim 6, and responsive to a request to invoke a function associated with a specific Overlay, includes the steps of:
testing the Overlay Control Structure to determine if the code portion of the specific Overlay is currently within the allocated code area; and
if the code portion of the specific Overlay is not within the allocated code area,
swapping the specific code portion from an associated allocated area within the expanded memory means to the allocated code area; and
updating the Overlay Control Structure to indicate that the code portion of the specific Overlay is currently within the allocated code area.

10. A method as claimed in claim 6 and further comprising the steps of:
invoking, with a first Overlay, a procedure associated with a second Overlay, the first Overlay code portion being within the allocated code area and the second Overlay code portion being within the allocated area within the expanded memory means;
the step of invoking a procedure including the steps of invoking an Overlay Load and Control

means;
modifying a first return address located on a first stack associated with the first Overlay to indicate a second return address, the second return address being a location associated with the Overlay Load and Control means;
placing information within a second stack, the information including an identification of the first Overlay and the first return address;
swapping the second Overlay into the allocated code area and executing the procedure; and
swapping the first Overlay back into the allocated code area based upon the information within the second stack.

11. A method as claimed in claim 10, wherein the step of modifying is accomplished by accessing the first stack within the allocated data area associated with the first Overlay.

12. A method as claimed in claim 8, wherein the step of testing includes an initial step of testing the Overlay Control Structure to determine if the function is associated with a swappable Overlay.

13. A method as claimed in claim 10, wherein the step of invoking an Overlay Load and Control means includes the steps of testing the Overlay Control Structure to determine if the function is associated with a swappable Overlay and testing the Overlay Control Structure to determine if an Overlay associated with the function is currently located within the allocated code area.

14. A method as claimed in claim 6 and including a step of storing data within the Overlay Control Structure, the step of storing data including the steps of:
for each of the Overlays,
storing an Overlay Status indicative of at least whether the Overlay is a swappable Overlay and whether the Overlay code portion is currently loaded within the allocated code area;
storing an Overlay Name;
storing an Overlay Initialisation Entry Point;
storing an Overlay Primary Entry Point;
storing an Overlay Termination Entry Point;
storing an Overlay Page Count indicative of the number of pages of the expanded memory means required to store the code portion; and
storing an Overlay Resident Data Segment Address.

15. A method as claimed in claim 6, wherein the steps of allocating are accomplished after a first step of accessing a mass storage means wherein the Overlays are stored.

16. A method as claimed in claim 15, wherein the steps of storing and the step of relocating are accomplished after a second step of accessing the mass storage means.

17. A method as claimed in claim 16, wherein the first and the second steps of accessing are accomplished by accessing .EXE-type files upon the mass storage means.

18. A method of swapping overlays in a data processing system including a paging memory, supporting a spectrum of system addresses to memory and the paging of information with respect to memory, the method comprising reserving a paging window in the spectrum of system addresses for paging overlay code and maintaining overlay data resident in segments of memory at other system addresses.

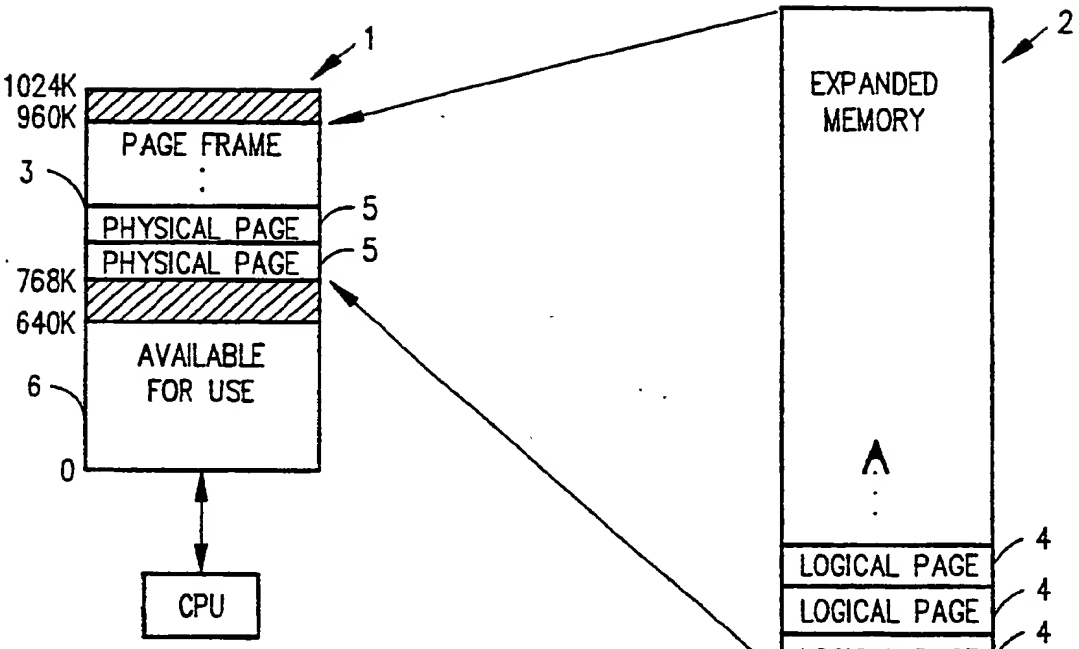

FIG. 1

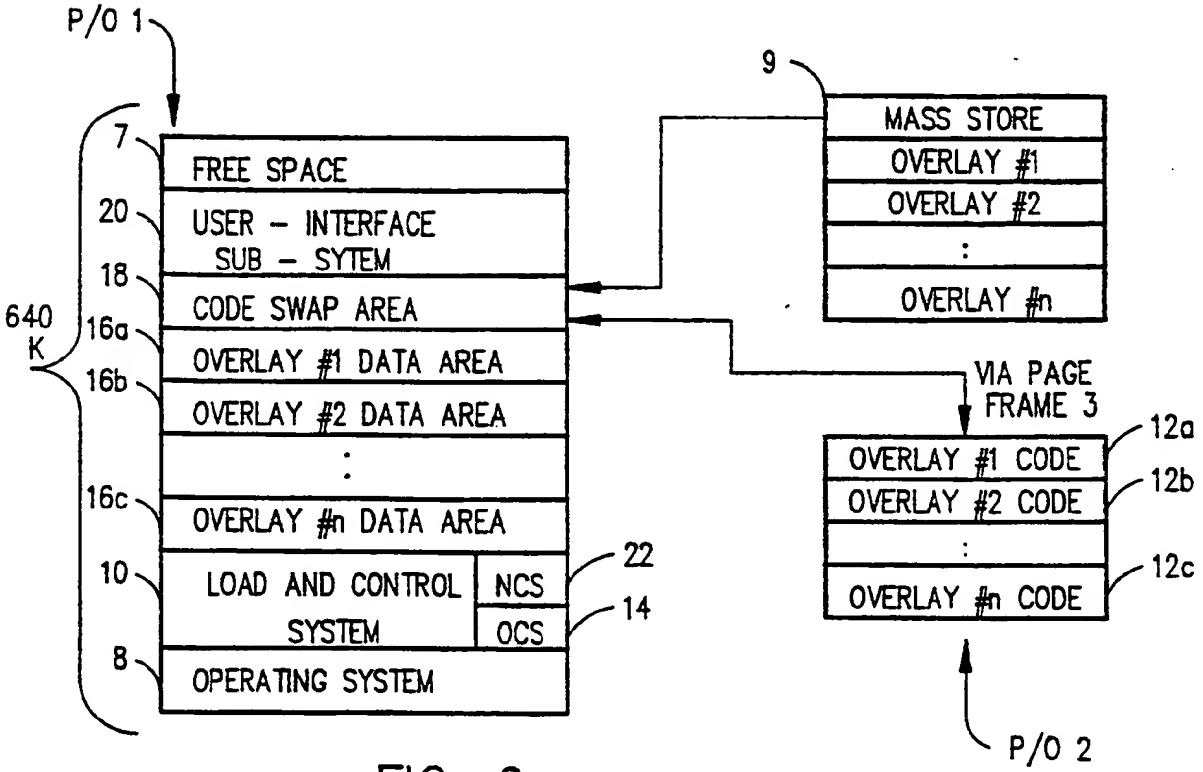

FIG. 2

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 403 124 A3**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: **90305974.9**

(51) Int. Cl.5: **G06F 12/08**

(22) Date of filing: **31.05.90**

(30) Priority: **16.06.89 US 367838**

(43) Date of publication of application:
19.12.90 Bulletin 90/51

(84) Designated Contracting States:
DE FR GB

(88) Date of deferred publication of the search report:
06.05.92 Bulletin 92/19

(71) Applicant: **International Business Machines Corporation**
Old Orchard Road
Armonk, N.Y. 10504(US)

(72) Inventor: **Sherman, Arthur Michael**
15920 La Escuela
Morgan Hill, CA 95037(US)
Inventor: **Walling, Lonnie Scott**
9780 Higway 9
Ben Lomond, CA 95005(US)

(74) Representative: **Mitchell, Allan Edmund et al**
IBM United Kingdom Limited Intellectual Property Department Hursley Park
Winchester Hampshire SO21 2JN(GB)

(54) Overlay swapping.

(57) Overlay swapping is enabled by differentiating Overlays each into a code portion and into a data portion, storing the data portions within a CPU directly-accessible memory, storing the code portions within a memory not-directly accessible by the CPU, and swapping a required code portion into the directly-accessible memory for execution by the CPU such that the swapped-in code portion has access to a plurality of the data portions stored within the directly-accessible memory.

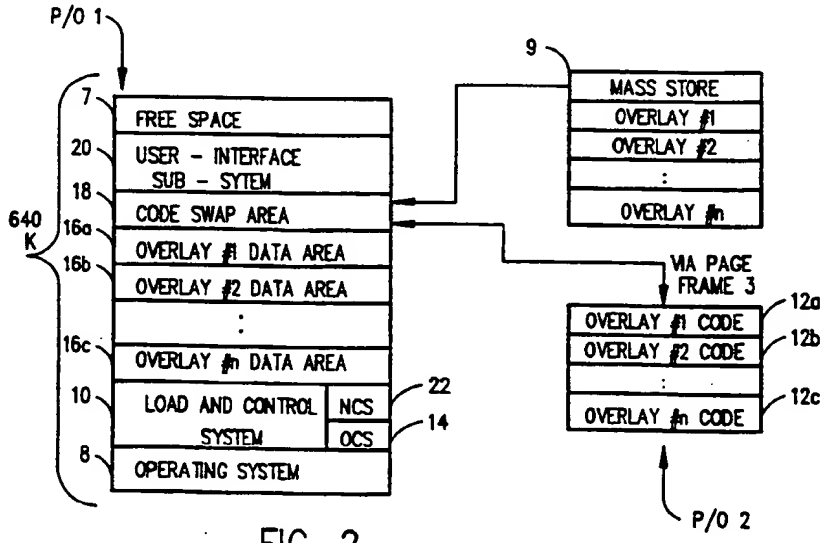

FIG. 2

Rank Xerox (UK) Business Services

European Patent Office

EUROPEAN SEARCH REPORT

Application Number

EP 90 30 5974

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int. Cl.5)
A	IBM TECHNICAL DISCLOSURE BULLETIN. vol. 28, no. 12, May 1986, NEW YORK US pages 5615 - 5620; 'Overlay Linker and Loader Process' * the whole document *	1-9,15, 16,18	G06F12/08
A	PATENT ABSTRACTS OF JAPAN vol. 012, no. 129 (P-692)21 April 1988 & JP-A-62 251 851 (SHARP) 2 November 1987 * abstract *	1,2,5,18	
A	PATENT ABSTRACTS OF JAPAN vol. 13, no. 070 (P-829)17 February 1989 & JP-A-63 257 034 (HITACHI LTD) 24 October 1988 * abstract *	1,2,5,18	
A	IBM TECHNICAL DISCLOSURE BULLETIN. vol. 16, no. 11, April 1974, NEW YORK US page 3626; DOGGETT: 'Reentrant Variable Initialization' * the whole document *	1,2,5,18	TECHNICAL FIELDS SEARCHED (Int. Cl.5) G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	25 FEBRUARY 1992	NIELSEN O.P.

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P0601)